

LIBRARY
OF THE
MASSACHUSETTS INSTITUTE
OF TECHNOLOGY

WORKING PAPER

ALFRED P. SLOAN SCHOOL OF MANAGEMENT

ON THE N-JOB, ONE-MACHINE, SEQUENCE-INDEPENDENT
SCHEDULING PROBLEM WITH TARDINESS PENALTIES:
A BRANCH-BOUND SOLUTION*

by

Joel Shwimer ✓

May 1970
465-70

MASSACHUSETTS
INSTITUTE OF TECHNOLOGY
50 MEMORIAL DRIVE
CAMBRIDGE, MASSACHUSETTS 02139



ON THE N-JOB, ONE-MACHINE, SEQUENCE-INDEPENDENT
SCHEDULING PROBLEM WITH TARDINESS PENALTIES:
A BRANCH-BOUND SOLUTION*

by

Joel Shwimer ✓

May 1970
465-70

*This paper is based substantially on my masters thesis. This copy is issued with the understanding that the paper is solely for the confidential use of the organization or individual to whom it is addressed.

Handwritten text, possibly a signature or date, located in the center of the page.

ON THE N-JOB, ONE-MACHINE, SEQUENCE-INDEPENDENT
SCHEDULING PROBLEM WITH TARDINESS PENALTIES:
A BRANCH-BOUND SOLUTION

by

Joel Shwimer

Massachusetts Institute of Technology

Abstract

An efficient branch-bound algorithm is presented for solving the n-job, sequence independent, single machine scheduling problem where the goal is to minimize the total penalty costs resulting from tardiness of jobs. The development of this algorithm and computational results are given for the case of linear penalty functions. The modifications needed to handle the case of non-linear penalty functions are also presented.

TABLE OF CONTENTS

	Page
I. The Problem.....	7
II. Other Research.....	9
III. Branch-Bound Algorithm.....	15
IV. Development of the Branch-Bound Algorithm.....	28
Stage 1.....	28
Stage 2.....	30
Stage 3.....	31
Stage 4.....	32
Stage 5.....	32
Stage 6.....	33
Stage 7.....	33
Stage 8.....	34
Stage 9.....	35
V. Computational Results for Final Algorithm.....	39
VI. Extensions and Future Work.....	44
Appendix I: Theorem A.....	48
Appendix II: Data Used in the Sample Problems.....	52
Bibliography.....	53

LIST OF TABLES

Table		Page
1	Summary of Developmental Stages.....	29
2	Developmental Stages Computational Results.....	37
3	Computational Results.....	42

LIST OF FIGURES

Figure		Page
1	Partial Solution Tree.....	25
2	Partial Solution Tree at Optimality.....	26
3	Complete Solution Tree.....	27

I. THE PROBLEM

The basic problem of sequencing n jobs on one machine has received much attention in the literature [1], [3], [9]. A number of important results and solution methods have been proposed for certain variations of this basic problem. One of the best-known of these results is that the mean flow time, mean waiting time, mean lateness, as well as a number of other "regular measures" of a schedule are all minimized by sequencing the jobs in order of non-decreasing processing time [1, p. 29]. However, no such simple results are available when certain alternative measures of schedule effectiveness are used.

The problem to be discussed in this paper is the scheduling of n jobs on a single machine where the goal is to minimize the total penalty costs resulting from the tardiness of the jobs. For each job i , there is a required processing time t_i , a due date d_i , and a penalty function P_i which is a function of the tardiness of job i . Although many of the results in this paper can be applied to the general case where the penalty function, P_i (tardiness of job i), takes on the value zero if the job is not tardy and is any non-decreasing function of tardiness, we will, at first, only consider the case of the linear penalty function:

$$P_i(d_i, C_i) = p_i \max(0, C_i - d_i)$$

where C_i denotes the time that job i is completed when the particular schedule being evaluated is used. All n jobs become available at time

zero and their processing times are independent of the sequence in which they are scheduled.

Although this problem is generally placed in the context of scheduling jobs on a machine, the situation being modeled occurs in a variety of different instances in the real world. Some examples of these occurrences include the scheduling of repairs at a garage, the scheduling of appointments in any single facility operation such as a doctor's office or a one-person travel agency, and even the scheduling of a student's time for homework assignments.

II. OTHER RESEARCH

This form of the basic problem was first presented by McNaughton [10], who argues for the desirability of a non-linear penalty function of tardiness, but deals only with the linear case. He proves the important theorem that, for the given problem, an optimal solution exists in which no job is split [10, p. 4]. Therefore we need only consider the various permutations of the n jobs to find the optimal schedule. However, even for modest-sized problems, complete enumeration is not computationally feasible since it requires the evaluation of $n!$ sequences (e.g. a 20 job problem requires the evaluation of more than 2.4×10^{18} sequences).

A number of approaches have been suggested for solving this problem without performing an explicit enumeration. McNaughton [10] presents one method and Schild and Fredman [12] generalize some of his results presenting another method. It has been shown [1, p. 48], however, that the latter does not insure an optimal solution. Schild and Fredman [13] also further generalize their method to consider the case of non-linear penalty functions. They develop a dynamic programming formulation although that term is never mentioned by them. No computational results are reported for any of these methods.

Held and Karp [6] present a dynamic programming formulation of this problem and Lawler [8] presents an equivalent formulation. These approaches require the consideration of 2^n possible subsets of the n jobs. The number of arithmetic operations is proportional to $n2^n$ and the storage required for simply the costs associated with each subset

of jobs is $2^n - 1$ numbers. This method, although much more efficient than complete explicit enumeration, is still computationally infeasible for even modest-sized problems (e.g. a 20 job problem requires the storage of more than a million numbers). Lawler does say that it is a "simple matter for a 15 job problem to be solved on an IBM 7090". However, no computation times are reported either by Lawler or by Held and Karp.

For the case of linear penalty functions, Lawler [8] also presents a linear programming formulation requiring $n+2T$ constraints, where T is the total required processing time for all n jobs. A simplex algorithm modified to handle a particular type of restricted entry constraint is required to solve this formulation.

It is also possible to formulate a much smaller mixed-integer linear programming formulation requiring only n^2 constraints, $3n$ continuous variables, and $n(n-1)/2$ zero-one variables. The current state of the art in solving such problems makes this formulation computationally infeasible for even modest-sized problems.

Elmaghraby [2] presents a network model which is equivalent to the earlier dynamic programming formulations. The cost-minimization objective function is translated into a shortest route problem. To create his network, Elmaghraby defines S_n to be the set of all jobs numbered in ascending magnitude of d_i . In general S_k denotes some subset of k jobs. If the set S_k represents the jobs to be performed, some job j in S_k must be performed last among the jobs constituting S_k . For each such job j , we would be left with a subset of $k-1$ jobs, denoted by S_{k-1}^j , which includes all of the jobs in S_k except for job j .

Thus, S_{k-1}^j is the subset of jobs still unscheduled after job j is scheduled last among the jobs in S_k . Hence there are k possible subsets S_{k-1}^j that can be derived from each S_k .

The network equivalent of this model is formed by representing the set S_n by a node and then joining it by n arcs to n nodes representing the sets S_{n-1}^j ($j=1, \dots, n$). This construction process continues until the nodes representing the empty set S_0 are reached. In general, the node representing some subset S_k is joined by k arcs to k nodes representing all of the possible subsets S_{k-1}^j which can be created from S_k . The length of the arc from the node representing some S_k to the node representing S_{k-1}^j is the penalty cost incurred by scheduling job j last among the jobs in S_k namely

$$\text{length} = p_j \max (0, T_k - d_j)$$

where

$$T_k = \sum_{i \in S_k} t_i \text{ is the completion time of the}$$

last job performed among the jobs of S_k if no idle time is allowed.

The goal is to find the shortest path from the node representing S_n to any node representing an empty set S_0 where the length of this path corresponds to the minimum possible total penalty cost. A schedule which achieves this minimum cost can be found by tracing back along the chosen path. Elmaghraby shows how branch-bound methods can

be applied to substantially reduce the required computation time as compared to the equivalent dynamic programming formulation.

In developing his branch-bound algorithm, Elmaghraby proves the following lemma:

If $\sum_{i \in S_k} t_i \leq d_{S_k}$ where $d_{S_k} = \text{MAX}_{i \in S_k} (d_i)$, then an optimal path

which passes through the node for this S_k must have the job in S_k with due date d_{S_k} , scheduled last among the jobs in S_k .

Elmaghraby also proves certain dominance relations between nodes in his branch-bound tree.

For the case of linear penalty functions, Elmaghraby outlines a six-step branch-bound algorithm using his lemma, certain dominance relations, and a lower bound at each node, representing a set of jobs S_k , computed by

$$LB(S_k) = f(S_k) + \text{MIN}_{j \in S_k} (p_j \max(0, T_k - d_j))$$

where

S_k is the set of unscheduled jobs;

$f(S_k)$ is the length of the shortest route from the node representing S_n to this node;

and

$$T_k = \sum_{i \in S_k} t_i$$

It would appear that certain steps (e.g., the dominance tests between nodes) would be quite hard to program efficiently for a computer. Large amounts of storage as well as a similar amount of computation would be required. No computational results were reported since Elmaghraby has not programmed this algorithm for a computer [4].

Emmons [5] considers the same basic problem with the objective of minimizing total tardiness rather than a weighted sum of job tardiness. He proves several theorems related to obtaining an optimal ordering of the jobs with respect to his objective function and then devises a branch-bound algorithm based on his theorems. This algorithm has also not been coded for a computer, but some of the hand-calculated examples (mainly 10 job problems) that he reports seem to indicate the possibility of promising computational results. Nevertheless, Emmons anticipates greater difficulty with problems in which the due dates tend to be larger than the processing times (which was not the case for 11 of his 12 examples). This, however, is probably the more typical situation among real-world problems.

Emmons also considers the more general objective of minimizing

$$\sum_{\text{all jobs}} g(x_i)$$

where

x_i is the tardiness of job i

g is any convex increasing function of $x_i \geq 0$

He generalizes several of his earlier results to this case, but the requirement that the penalty functions be the same for each job appears to be a severe limitation.

III. BRANCH-BOUND ALGORITHM

The primary purpose of this paper is to develop and present a branch-bound algorithm for solving the n -job, one-machine, sequence-independent scheduling problem where the objective is to minimize the sum of the incurred tardiness penalties. The work of Elmaghraby [2] provides a good framework within which to work; his shortest route formulation is the underlying structure for this algorithm.

The branch-bound algorithm will now be listed followed by a discussion of the approach used and justification for some of the steps employed:

Step 0. Initialize.

- a. Order the jobs by increasing d_i , breaking ties by placing the job with the larger p_i first, and breaking remaining ties by placing the job with the smaller t_i first. (Any remaining ties occur because the two jobs have exactly the same characteristics and these ties can be broken arbitrarily.)
- b. Test each pair of jobs i, j ($i < j$) by Theorem A: If $p_i \geq p_j$, $d_i \leq d_j$, and $t_i \leq t_j$, then save the information that we will only consider schedules with job i preceding job j . (A proof of Theorem A is included in Appendix I.)
- c. Create root node and put it on node list. Perform the bookkeeping initialization.
- d. Search for a good initial problem upper bound by considering the n schedules formed by taking the jobs as

ordered in step 0.a except for job i ($i=1,\dots,n$) which is scheduled last. Save the best of these n schedules and the associated total penalty cost as the current "best".

Step 1. Choose a Node for Expansion

Consider the last node placed on the node list. If it has been already expanded, or if its value is not less than the current "best" (i.e. it does not have the potential to produce a better schedule than the current best), drop it from the node list and start step 1 again.

When the node list is empty, stop, since the current best schedule has been proved optimal.

Otherwise expand the last node on the node list.

Step 2. Expand Chosen Node

To expand a node, say node k , consider the set, S_k , of all jobs as yet unscheduled at that node.

- a. If S_k consists of only one job, m , find the total penalty cost of the schedule formed by placing job m first, by adding $p_m \max(0, d_m - t_m)$ to the incurred cost at node k . If this new cost is less than the current "best", save this schedule and the associated total penalty cost as the new "best". Otherwise the old "best" remains. In either case, drop node k from the node list and go to step 1.
- b. If S_k consists of more than one job, test to see if the

total processing time of all as yet unscheduled jobs is less than or equal to the due date of the job, j , with the latest due date among unscheduled jobs. If this condition holds, only consider creating one successor node to node k by scheduling job j last among the jobs in S_k . Then go to step 2.d. This is the use of Elmaghraby's lemma.

- c. Otherwise, we must consider creating one successor node to node k for each job in S_k that can be scheduled last among the jobs in S_k . Application of theorem A (step 0.b) may have determined that job i must precede some job j . Thus if both job i and this job j are in S_k , we need not consider scheduling job i last among the jobs in S_k .
- d. Evaluate each potential successor node to node k (i.e. the scheduling of job j last among the jobs in S_k) in the following fashion:

$$\text{VALUE} = \text{INCURRED COST AT NODE } k$$

$$+ p_j \max (0, d_j - T_k)$$

$$+ \min_{\substack{i \in S_k \\ i \neq j}} \{ p_i \max (0, d_i - T_k + t_j) +$$

$$\begin{array}{lcl}
 & \text{tardiness of job } h & \\
 + \text{MAX} & \text{in a schedule formed} & \\
 h \in S_k & \text{by ordering the jobs} & \{ \text{MIN } (p_g) \} \\
 h \neq j & h \in S_k, h \neq j, h \neq i \text{ as in} & g \in S_k \\
 h \neq i & \text{step 0.a} & g \neq j \\
 & & g \neq i
 \end{array}$$

where

j denotes the job being scheduled to create

this node as a successor to node k ,

S_k denotes the set of unscheduled jobs at node k ,

$T_k = (\sum_{i \in S_k} t_i)$ is the processing time of all

jobs unscheduled at node k .

If the value of the node created by scheduling job j is not less than the current "best", this node can be discarded without ever being added to the node list (i.e. without ever creating it).

Otherwise, the incurred cost for this new node should be calculated as follows:

INCURRED COST = INCURRED COST AT NODE k

$$+ p_j \max (0, d_j - T_k)$$

e. Now order all of the nodes created in step 2.b or 2.c but not discarded in step 2.d in order of decreasing

VALUE and place them on the node list as nodes $k+1$,
 $k+2$

f. Mark node k as expanded and go to step 1.

The above procedure creates the same basic tree as in Elmaghraby [2] (described in Section II of this paper); however, the means for bounding the tree, as well as many other steps in the algorithm, are quite different from his procedure. The algorithm outlined above is a "restricted flooding" branch-bound method¹ since the node to be expanded is always on the lowest level (i.e. the farthest from the root node) on the tree. The partial schedule having the minimum lower bound on the total penalty cost (VALUE) is chosen to be the one next evaluated, from among those on the tree having the most jobs already scheduled. Proceeding this way results in more complete schedules, hence more feasible solutions, to be reached sooner. Therefore, if the algorithm is stopped before optimality has been proven, there is a greater chance of having reached a better solution.

Step 2.b is the test indicated by Elmaghraby's lemma, stated in Section II of this paper. In step 2.c, it is not always necessary to consider all still unscheduled jobs because of the application of Theorem A. This theorem is an extension of Emmons's theorem [5, p. 703] which states:

For any two jobs i and j , if 1) $t_i \leq t_j$, 2) the jobs in set

B precede job j in an optimal schedule, and

¹This term was first heard by the author in a seminar on Tree-Search Methods taught by J. Pierce in Spring, 1969 at M.I.T.

3) $d_i \leq \max (\sum_{k \in B} p_k + p_j, d_j)$; then only schedules with job i preceding job j need be considered in searching for the optimal.

Theorem A extends this result to the case where the penalty function for each job can be different. It also presents the static version by removing any consideration of a particular set of jobs B which must precede job j . A proof of Theorem A is presented in Appendix I.

The method of calculation of the lower bound on the total penalty cost used in step 2.d combines several results proven by different people. To find this lower bound value for a successor node to node k , Elmaghraby uses:

$$\begin{aligned} \text{VALUE} = & \text{INCURRED COST AT NODE } k + p_j \max (0, T_k - d_j) \\ & + \min_{\substack{i \in S_k \\ i \neq j}} (p_i \max (0, d_i - T_k + t_j)) \end{aligned}$$

This lower bound includes the smallest possible penalty cost incurred by scheduling some job last among the remaining unscheduled jobs after job j is scheduled last among the jobs in S_k . It is a one-step, look-ahead procedure. The lower bound calculated in step 2.d goes still further. Let job j denote the additional job to be scheduled to create this node. The one-step look-ahead procedure considers the scheduling of some job, say job i , last among the jobs in S_k excluding

job j . However other jobs in S_k may still remain unscheduled. We should consider any additional penalty cost which must be incurred because of these jobs.

Jackson [7] has shown that the maximum tardiness in a one-stage process can be minimized if the jobs are ordered by increasing due date. So sequencing the jobs in S_k , other than jobs j and i , gives us a lower bound on the tardiness that must still be incurred. Since we seek weighted tardiness rather than simply tardiness, a lower bound on the weighted tardiness that must be incurred can be found by using the minimum of the p_g for this set of jobs. Taking the minimum of the sum of these two additional penalty costs (over all jobs i in S_k excluding job j) gives us a tighter bound on penalty cost than was obtained in the one-step, look-ahead method.

Storage requirements for this method are quite low. Since the above-described restricted flooding technique is used for choosing the node to be expanded, a maximum of $(n^2+n)/2$ nodes will have to be maintained at any given time (where n is the number of jobs in the problem). In fact, it is exceedingly unlikely that this maximum will ever actually be needed. To fully specify a node, at most the following seven numbers are needed:

- 1) a node identification number (unique for each node that is currently being maintained);
- 2) the identification number of its parent node;
- 3) the job which was scheduled to create this node from its parent node;

- 4) the total processing time required for all unscheduled jobs (T_k in above discussion);
- 5) the cost incurred by those jobs already scheduled upon reaching this node;
- 6) a lower bound (VALUE) on the cost of all solutions passing through this node; and
- 7) a marker to indicate the status of this node (i.e. expanded or not).

In the current implementation of the algorithm, six numbers are stored for each node (the seventh number, item one, being implied by the search routine). Therefore, a maximum of $3(n^2 + n)$ numbers must be saved at any one time (e.g. for n equal 20, this is only 1260; and for n equal 40, this is still only 4920). In fact, this storage requirement could be reduced by 50% since items four and five could be easily calculated by tracing up the tree, and item seven could be stored simply negating item two or three to indicate an expanded node. However, the increased computation time needed to implement this reduced storage scheme would be significant.

The branch-bound algorithm described above has been programmed in Fortran and fully tested. A discussion of the steps in its development are given in Section IV of this paper to show the effect each improvement had on the over-all solution method. Some computational results for the final algorithm will be presented in Section V.

Before presenting the various stages in the development of the final algorithm, an example will be presented to show how the tree develops as the algorithm proceeds towards finding the optimal solution and proving its optimality. The example, which will later be referred to as Problem 3, is a 10 job problem with the minimum total penalty cost of 27. The actual data to define this problem is included in Appendix II (as Problem 3).

There are $10!$ (i.e. 3,628,800) feasible schedules if idle time and job-splitting are not permitted. Application of Theorem A enables us to determine that we need consider only those schedules in which job 1 precedes job 4, job 2 precedes job 7, job 3 precedes jobs 4, 7 and 9, job 5 precedes jobs 7 and 9, and job 8 precedes job 9. This reduces the number of feasible schedules that must be considered to only 127,140 (a reduction of over 96%).

In performing step 0.d which searches for a good initial problem upper bound, we find that the completely due-date ordered schedule is, in fact, the best of the ten schedules considered. It has a total penalty cost of 29. Therefore we never have to create any node which has a VALUE of 29 or larger.

Figure 1 shows the tree which has been created as the algorithm proceeded to reach its first completed schedule. This schedule, (1, 2, 3, 5, 4, 6, 7, 8, 9, 10), has a cost of 28, which is, of course, less than 29. Only 16 nodes have been created and a great majority of the feasible schedules have already been eliminated from consideration.

Figure 2 shows the tree when the next complete schedule is reached. This new schedule, (1, 2, 3, 5, 4, 6, 8, 9, 7, 10), costs only 27 and will turn out to be an optimal one. Note how large portions of the tree have already been discarded. A total of 32 nodes have been created so far. Only four additional nodes will have to be created to prove optimality. Figure 3 shows the entire tree of 36 nodes used in solving this problem. The entire tree was never held in the computer at one time. The generation and discarding procedure required a maximum of 16 nodes to be stored in the computer at any one time (less than 30% of the maximum possible number of nodes that might have been needed at any one time).

Figure 1

Partial Solution Tree

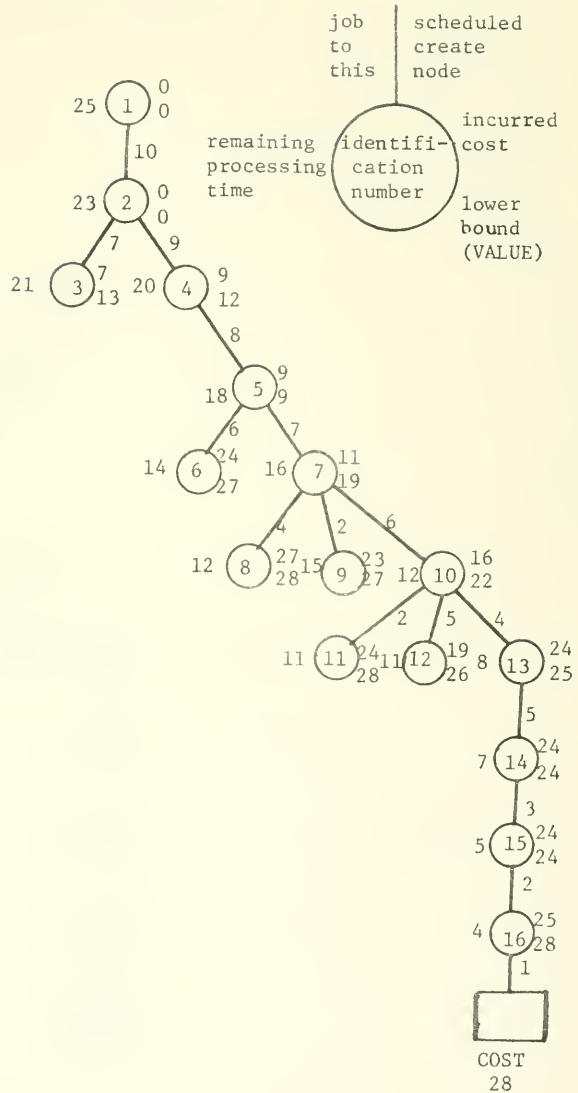


Figure 2

Partial Solution Tree
at Optimality

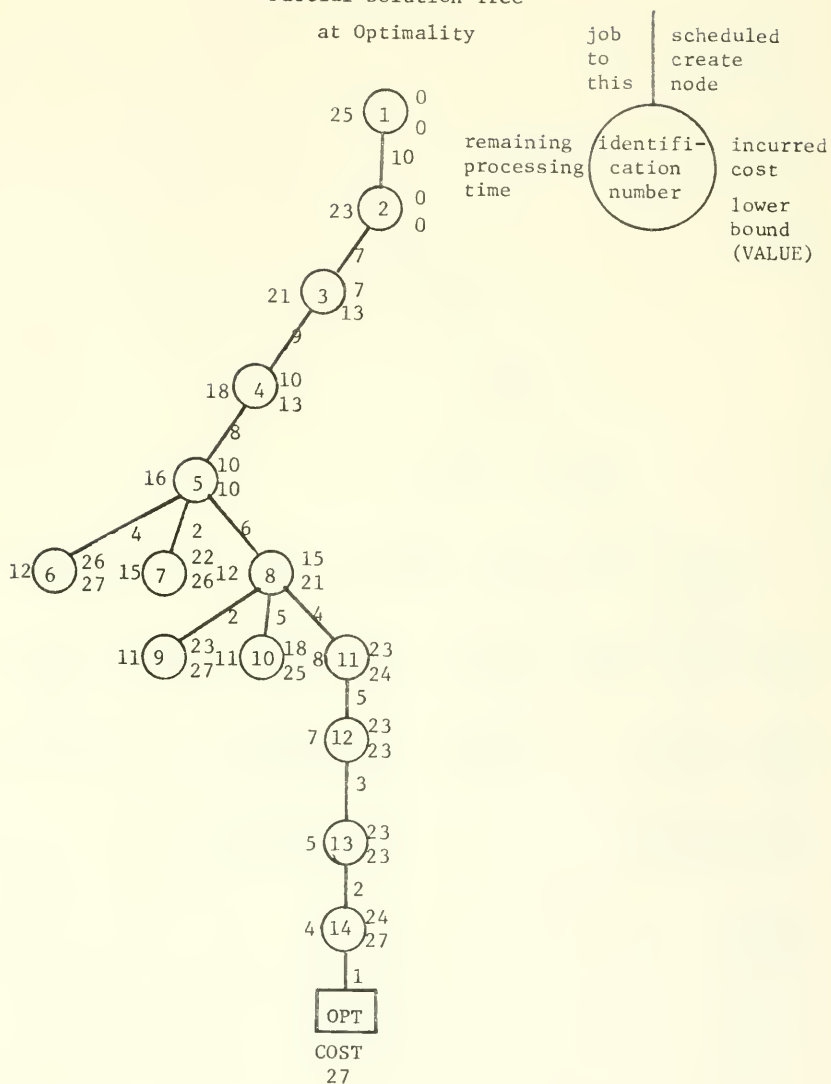
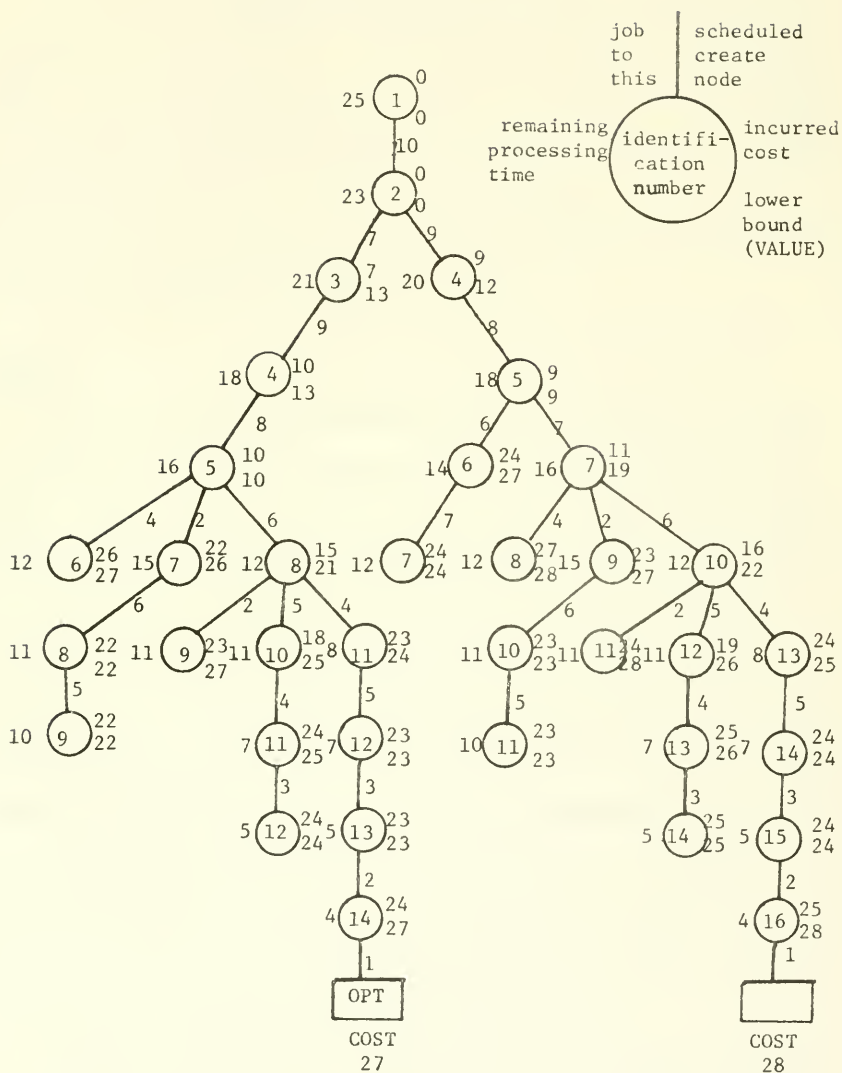


Figure 3

Complete Solution Tree



IV. DEVELOPMENT OF THE BRANCH-BOUND ALGORITHM

This section of the paper will describe the nine stages of the development of the algorithm. This discussion should be particularly useful to people who might, in the future, use branch-bound techniques for solving combinatorial optimization problems. Each stage was fully programmed and tested on a computer in an attempt to see how each modification to the algorithm would affect its ability to solve the problem. The stages are summarized in Table 1.

STAGE 1

The initial stage concentrated on obtaining a method for solving the problem, with little thought given to over-all efficiency of the method. Hence the scheme used was one of simple implicit enumeration based on a due date ordering. First, the jobs are arranged in order of increasing due date (job 1 has the earliest due date and job n, the latest). Ties are broken as indicated in Step 0.a of the final branch-bound algorithm. An initial "problem upper-bound" is obtained by computing the total penalty cost incurred by this "due-date-ordered" schedule. Then using the root node (node 1 representing all jobs unscheduled) as the starting point, n successor nodes are created corresponding to scheduling each of the n jobs last. Nodes which represent partial schedules with incurred costs larger than or equal to the current problem upper bound are never created.

The algorithm proceeds by considering the last node to have been created which has not yet been discarded. If this node has been

Table 1

SUMMARY OF DEVELOPMENTAL STAGES

<u>Stage</u>	<u>Description</u>
1	Basic enumeration scheme based on due-date ordering
2	Stage 1 with Elmaghraby's lemma
3	Stage 2 with Elmaghraby's calculation for lower bound at each node
4	Stage 2 with problem upper-bound calculations for all nodes which are immediate descendants of the root node
5	Stages 3 and 4 combined
6	Stage 5 with step 2.d lower bound calculation for all nodes
7	Stage 6 with more sophisticated "restricted flooding" technique choosing the node with the smallest lower bound at the lowest level to be explored next
8	The same as Stage 7 except for a new bookkeeping method for knowing unscheduled jobs at a node
9	Stage 8 with Theorem A included

expanded it is now discarded and the next most recently created node is considered. If the incurred cost for this node is less than the current problem upper bound it is expanded in the same manner as described above for the root node (at most k successor nodes for a node corresponding to a set of k unscheduled jobs); otherwise this node is discarded and the next most recently created node is considered. Whenever a complete schedule is reached, it is retained only if it is better than the best previously found solution (the problem upper bound being revised accordingly). This naive, branch-bound procedure will eventually find an optimal solution and prove its optimality by implicitly considering every possible schedule.

STAGE 2

It is obvious that other methods for choosing the node to explore next and for calculating a lower bound on the cost of all schedules passing through any node are both possible and desirable. The second stage includes the use of Elmaghraby's lemma. When any node is being expanded, a check is made to see whether the job with the latest due-date among the unscheduled jobs can be placed last among this set without incurring any penalty cost. If this is true, only the one successor node has to be created; thus the number of nodes that are created and later explored is greatly reduced. It is not possible for the one successor node to overlay its parent because, in later passes through this part of the tree, certain necessary information (e.g. which jobs have been already scheduled) would be lost. Under a

different storage scheme for the tree, the reduction in required storage might prove both feasible and beneficial, however using the currently implemented scheme, even the benefits would be so minimal as to rule out the desirability of such an idea.

In most cases, a significant reduction in required computation was realized because the added computation needed for the check is minimal while the reduction in number of created nodes can be quite large.

STAGE 3

The next step was to improve the lower bound calculation made at each node. As a first step in this direction, the one-step, look-ahead procedure suggested by Elmaghraby was used. Thus, for a node being created by scheduling job j last among the set, S_k , of unscheduled jobs, a lower bound may be calculated from:

VALUE = INCURRED COST AT PARENT NODE

$$\begin{aligned}
 &+ p_j \max (0, d_j - T_k) \\
 &+ \min_{\substack{i \in S_k \\ i \neq j}} [p_i \max (0, d_i - T_k + t_j)]
 \end{aligned}$$

$$\text{where } T_k = \sum_{i \in S_k} t_i$$

Again, at the cost of some minimal increased computational effort, this improved lower bound should help to reduce the number of nodes

that must be created and later explored. Testing showed the number of nodes created always to be reduced; however, because of the increased computational effort in calculating the lower bounds, the total computation time sometimes increased slightly.

STAGE 4

In this stage, the improved lower bound calculation of STAGE 3 was dropped and an attempt to obtain a better initial problem upper bound was included. Rather than considering only the due-date-ordered schedule, the n schedules formed by due-date-ordering, except for job i ($i=1, \dots, n$) which was performed last, were evaluated. The best of these n schedules was retained as the "current optimal" solution. The algorithm proceeded exactly as did the STAGE 2 method after performing this modified method for obtaining an initial problem upper bound. Reductions in the number of nodes created and in the total computation time were almost always obtained. Having the lower "current best cost" allowed many nodes not to be created. This resulted in the saving of computation time.

STAGE 5

Since the modifications for both STAGE 3 and STAGE 4 appeared to be promising, both were included to produce STAGE 5. Thus, attention was directed to finding a better initial problem upper bound and to finding larger lower bounds at the various nodes being created. Moreover, no node would ever be created unless its lower bound was less

than the current problem upper bound. This combination, in almost all cases, proved to be better than either STAGE 3 or STAGE 4 by itself in reduction of both nodes created and computation time.

STAGE 6

A further improvement was made to the lower bound calculation in creating STAGE 6. This calculation is the one listed as step 2.d of the **final** algorithm. Again, at the cost of increased computation in the lower bound calculation, many nodes could be discarded before creation thereby saving future exploration of these nodes and their possible successors. In most cases improvement was realized. However with certain data, the increased computation time needed to calculate the tighter lower bounds was greater than the reduction in computation time due to fewer nodes being created.

STAGE 7

In the previous stages, attention had been paid only to improving the calculation of the bounds, both the initial problem upper bounds and the lower bounds at each node being created. In this stage, effort was directed towards improving the choice of which node to expand next. The use of a more sophisticated "restricted flooding" technique replaced the choice according to the due-date-ordered implicit enumeration scheme. If all nodes representing subsets of k unscheduled jobs are said to be at "level k ", restricted flooding requires that the node chosen for exploration be in the (numerically) lowest available level.

Thus, if possible, the search proceeds to a lower level on each step. The change that was implemented in this stage was to choose, as the node to be expanded, the one on the lowest level which had the smallest lower bound.

The revised branching procedure was easily placed into the framework used for the previous stages. When a node is being expanded, instead of creating the successor nodes in order of increasing due date, all of these nodes can be created and then placed on the tree in order of decreasing lower bound. Thus the node placed on the tree last will be the one on the lowest level with the smallest lower bound (i.e. the one to be explored next). STAGE 7 includes this restricted flooding rule for branching in conjunction with all of the improved bounding rules incorporated in previous stages.

Testing showed that this improvement yielded better reductions in computation time than any other improvement except for the use of Elmaghraby's lemma in STAGE 2. Not only was overall computation time reduced, but even greater reductions were obtained in the time to find the optimal solution. Thus, if the algorithm is stopped before optimality has been proved, there is a greater chance of having reached a better, if not in fact an optimal, solution.

STAGE 8

A new bookkeeping method for keeping track of which jobs are still unscheduled at any node was adopted. Rather than searching up the tree each time this knowledge was required, a scheme for continually

updating this information as the search moved about the tree was implemented. This had no effect on the number of nodes created or the order of their exploration, but did result in a significant reduction of the required computation time.

STAGE 9

The final improvement in the branch-bound algorithm was the inclusion of the results of Theorem A.

Theorem A:

If $t_i \leq t_j$, $d_i \leq d_j$, and $p_i \geq p_j$ then there exists an

optimal solution in which job i precedes job j .

Theorem A can impose restrictions (care must be taken not to permit conflicting restrictions such as job i precedes job j and job j precedes job i) which help to greatly limit the number of schedules to be considered.¹

Each of the above stages was programmed in Fortran and tested on an IBM 1130 computer. The results of the test runs are included in this paper as Table 2. Problem 1 is the example from Elmaghraby [2],

¹Use of this theorem restricts the $7! = 5040$ possible schedules in Elmaghraby's example to only 252. When this theorem is used on the data for Emmon's example 1, the $10! = 3628800$ possible schedules are reduced to only 1836. Although this is, perhaps, an extreme example it does illustrate the potential benefits to be derived by using this theorem.

Problem 2 is example 1 from Emmons [5], and the other four problems were created somewhat arbitrarily by hand for the purpose of testing the computer programs. The actual data for all six problems is included as Appendix II.

These computational results are being included to demonstrate the effect of each improvement in the algorithm on its solution ability. The computer time required is probably the most important measure of effectiveness. It is apparent that the later stages dominate the earlier ones, particularly for the larger problems.

Table 2

Developmental Stages Computational Results

Problem	1	2	3	4	5	6
No. of Jobs	7	10	10	15	20	20
Maximum Possible Solutions	5040	3228800	3228800	$>10^{12}$	$>10^{18}$	$>10^{18}$
Cost of Optimal Solution	50	1211	27	33	58	61

The entries in the table below have the following form:

computation time in hours on IBM 1130 computer

nodes created in
reaching optimal
solution / nodes created in
reaching optimal
solution and proving
optimality

Stage 1	.002 56/56	.038 5588/6311	.003 103/220	.052 4080/6378	*	*
Stage 2	.002 46/46	.041 5523/6246	.002 62/99	.006 507/526	.040 3930/4212	.098 9539/10674
Stage 3	.001 39/39	.026 1596/1726	.002 55/91	.005 460/478	.044 3676/3942	.101 8322/9312
Stage 4	.001 1/22	.038 5157/5880	.002 62/99	.004 421/440	.019 1629/1911	.093 8855/9990
Stage 5	.001 1/16	.023 1456/1586	.002 55/91	.004 377/395	.020 1511/1777	.095 7690/8680
Stage 6	.001 1/8	.030 515/535	.002 41/48	.004 237/240	.019 1121/1291	.085 5509/6194
Stage 7	.001 1/8	.011 68/112	.002 36/48	.002 103/124	.008 148/427	.048 118/3458

Table 2
(continued)

Stage 8	.001 1/8	.011 68/112	.001 36/48	.002 103/124	.005 145/424	.013 118/3458
Stage 9	.001 1/8	.002 38/45	.001 32/36	.002 92/113	.006 111/225	.013 56/751

* Run stopped after 1.0 hour.

V. COMPUTATIONAL RESULTS FOR FINAL ALGORITHM

In order to provide a more extensive test for the final algorithm (Stage 9), a program was set up to generate sets of data drawn randomly from prescribed distributions. The processing times, t_j , were drawn either from a uniform distribution with range one to ten or from a triangular distribution with the same range. The due dates, d_j , were drawn from a uniform distribution with range t_j (so that no job would be late if it were started at time zero) to $5.5N$ (where N was the number of jobs in the problem and 5.5 was the expected mean of the distribution of processing times due to the integerization of the times). The penalty function coefficients, p_j , were uniformly distributed integers between one and five. All of the generated data was kept as integers. This method was used for data generation because it provided what appeared to be somewhat realistic data.

The program was used on an IBM 360/65 computer where system routines enabled accurate timing of the computation (to the nearest hundredth of a second). Fortran IV, H-level (optimization level 2) was used for the compilation of the program. It was found that problems ran approximately 48 times faster on this system than they had on the IBM 1130 computer used for the initial testing and stage comparison runs.

Data was created for sixty test problems in each of the following four categories:

- 10 job problems with uniformly distributed processing times;
- 10 job problems with triangularly distributed processing

times;

20 job problems with uniformly distributed processing times;

and

20 job problems with triangularly distributed processing times.

The computational results for these 240 problems are summarized in Table 3. From these results it appears, as was expected, that problems having processing times drawn from a uniform distribution are somewhat easier to solve than those with triangularly distributed processing times. The median solution time for all of the 10 job problems was 0.07 seconds; for all of the 20 job problems, the median solution time was 0.29 seconds. These results show that the algorithm is certainly computationally feasible for such modest-sized problems. Preliminary investigation indicates that somewhat larger problems can still be solved within quite short periods of computer time. For 30 job problems, the median solution time for a sample of thirty problems (half having uniformly distributed processing times and half having triangularly distributed processing times) was 4.13 seconds.

Computational experience has shown (see Table 3) that some small fraction of the problems may require a long time to solve (i.e. much greater than the median solution time for that size problem). Because of this, it is important to see where in the solution process the optimal schedule is actually found. In about 45% of the 10 job problems and 15% of the 20 job problems, an optimal schedule was found in step 0.d of the algorithm (i.e. during the search for an initial

problem upper bound). Problems in which the optimal solution was found either in step 0.d of the algorithm or at the first complete schedule reached in searching down the tree comprised over 80% of the 10 job problems and over 45% of the 20 job problems. These results help to explain the skewness of the solution time distributions.

It is also interesting to observe how much time is used in finding an optimal schedule as compared to how much time is then used in proving optimality. The average fraction of the solution time spent in finding the optimal solution varied between 65 and 70 percent for the four sets of problems that were solved. However, for the larger (i.e. 20 job) problems, less than 60% (on the average) of the solution time was spent in finding the optimum for those problems which required more than the median time to solve completely. These are the problems for which it might be necessary to stop the algorithm short of completion and fortunately, these are also the problems in which the optimal solutions are found sooner relative to the total required solution time.

Table 3

COMPUTATIONAL RESULTS

Number of jobs to be scheduled	10	10	10	10
Distribution of t_j	uniform	triangular	uniform	triangular
Number of problems run	60	60	60	60
Number of problems solved	60	60	60	59*
Median solution time	.08	.07	.28	.35
Interquartile range for solution times	.04	.04	1.00	7.88
Range for solution times	.03-.19	.03-.35	.08-89.87	.09->600.
Mean solution time	.085	.088	5.897	
Standard deviation of solution times	.035	.051	16.107	
Skewness** of solution time distribution	1.231	2.649	3.813	
Kurtosis*** of solution time distribution	4.339	12.848	17.771	
Fraction of problems where the optimal solution was found at an immediate successor of the root node	.467	.450	.167	.150

Table 3

(continued)

Fraction of problems where the optimal solution was found at an immediate successor of the root node or at the first completed schedule found searching down the tree	.867	.817	.500	.417
Average fraction of the solution time used to find the optimal solution (for problems with solution times below median solution time)	.645	.635	.817	.759
Average fraction of the solution time used to find the optimal solution (for problems with solution times greater than median)	.659	.743	.591	.569
Average fraction of the solution time used to find the optimal solution (for all problems in the group that were solved)	.652	.689	.704	.667

All times are given in seconds of IBM 360/65 CPU usage.

* One problem was stopped after ten minutes of computer time.

** Skewness being defined as μ_3/σ^3 .

*** Kurtosis being defined as μ_4/σ^4 .

VI. EXTENSIONS AND FUTURE WORK

A branch-bound algorithm has been presented to solve the problem of scheduling n immediately available jobs for a single machine where the goal is to minimize the total penalty cost resulting from linear penalty functions of the tardiness of the jobs. This work can be extended in several directions.

One extension which can be easily handled by the current algorithm is the inclusion of pairwise precedence constraints. It might be desirable to specify that job b must precede job c and job e must precede job d . The mechanism for handling such constraints has already been incorporated to implement the results of Theorem A. The only modification to the algorithm that is needed would be to insure that Theorem A does not "create" any pseudo-precedence constraints which are inconsistent with those specified in the problem (e.g. if job b must precede job c , and job c must precede job d , we cannot allow Theorem A to specify that we will only consider schedules in which job d precedes job b).

Another extension would be to allow for non-linear penalty functions. The following modifications would be necessary. Theorem A would have to be generalized in the following manner:

Theorem A (generalized):

If $t_i \leq t_j$, $d_i \leq d_j$, and $P_i(x) \geq P_j(x)$ for all x ,

$0 \leq x \leq T - d_i$, then there exists an optimal solution in which job i precedes job j (where $P_i(x)$ is the penalty function of

tardiness for job i , and T is the total processing time required by all n jobs).

The incurred cost calculation in step 2.d would have to be changed to:

$$\text{INCURRED COST} = \text{INCURRED COST AT NODE } k + P_j(\max(0, d_j - T_k))$$

The lower bound calculation, step 2.d, would also have to be changed to:

$$\begin{aligned} \text{VALUE} = & \text{INCURRED COST AT NODE } k + P_j(\max(0, d_j - T_k)) \\ & + \min_{\substack{i \in S_k \\ i \neq j}} [P_i(\max(0, d_i - T_k + t_j))] \\ & + \min_{\substack{g \in S_k \\ g \neq j \\ g \neq i}} P_g(\text{MAX} \begin{array}{l} \text{tardiness of job } h \text{ in a schedule} \\ \text{\{formed by ordering the jobs in } S_k, \}} \\ h \neq j, h \neq i \text{ as in step 0.a} \end{array}) \end{aligned}$$

For linear penalty functions, this bound reduces to the one presented in the above algorithm. These are the only changes needed to handle non-linear penalty functions provided they are non-decreasing functions of tardiness which take on the value zero if the tardiness equals zero.

It is quite possible to suggest ways to modify, and hopefully improve, the branch-bound algorithm. Many of these have already been considered and will be now discussed briefly. It may be beneficial

to expend a greater effort in obtaining a better initial problem upper bound. Any number of heuristics could be used for this purpose, and some have been tested. Even though the extra computational effort could be kept to a minimum, the net effect on total computation time might even be detrimental. The current method yields very good results. In fact, many problems become optimal at the initial step or at the first complete schedule found on the tree. Thus, on the average, very few nodes would be eliminated.

The use of a heuristic to generate complete solutions at nodes other than the root node may prove helpful in the early stages of the search. However, in the later stages, the only effect of such a technique would be to increase computation time. The possibility of performing a complete enumeration to expand a node after some level in the tree has been reached was also considered. However, the use of Elmaghraby's lemma, Theorem A, and the lower-bound calculations seem to be more efficient than enumeration with regard to computation time.

The use of dominance relations between nodes, as suggested by Elmaghraby [2], is another possible area for exploration. It is clear that such relations could greatly reduce the necessary amount of searching in the lower levels of the tree. However, great care would have to be used in performing such tests. The vast number of nodes that might be created would require a large amount of storage. Also, the tests would involve searching through a long list of previously created nodes which might prove to be very time consuming. One way to include these tests would be to save all nodes created at some

particular level of the tree (say about two-thirds of the way down) and then to apply the dominance tests only at that level. As problems become larger, the potential gains become greater, but so do the potential storage and computation time requirements.

One other set of possibilities is to incorporate more results of other researchers (e.g. Emmons [5] or Schild and Fredman [12]). Most of these require working from both ends of the schedule at the same time. This would necessitate major changes to the current method of maintaining the tree and many of the current "time-saving" features would be lost. Other results, perhaps even a dynamic version of Theorem A, are indeed possible, but the potential gains would in most cases compare unfavorably with the required extra computation time.

It is clear that a reduction in computation time could easily be obtained by reprogramming the algorithm in an assembly language rather than in Fortran. Also certain tests for optional output (e.g. at the creation of each node) could be removed.

Appendix I

Theorem A:

For any two jobs J_i and J_j such that $0 \leq t_i \leq t_j$, $0 \leq d_i \leq d_j$, and $p_i \geq p_j \geq 0$, there exists an optimal schedule in which job J_i precedes job J_j .

Proof:

Consider any schedule in which job J_j precedes job J_i . We must show that the total penalty cost is either decreased or left unchanged by interchanging these two jobs. First, observe that any jobs which are scheduled before job J_j or after job J_i are not affected by this interchange. Also note that any jobs scheduled between jobs J_j and J_i are advanced in time by an amount $t_j - t_i \geq 0$ thereby insuring that the penalty cost arising from these jobs has been either reduced or left unchanged. Thus we need consider only the change in penalty costs incurred by jobs J_i and J_j themselves.

Denote by K , the time at which job J_j is finished in the original schedule; and by L , the time at which job J_i is started. Thus $K \leq L$. The penalty cost, P' , due to these two jobs in the original schedule can be represented as:

$$P' = p_j \max(0, K - d_j) + p_i \max(0, L + t_i - d_i)$$

The penalty cost, P'' , due to these two jobs after they have been interchanged can be represented as:

$$P'' = p_j \max (0, L+t_i-d_j) + p_i \max (0, K-t_j+t_i-d_i)$$

Therefore we must show that

$$\Delta P = P'' - P' \leq 0 \quad \text{for all } 0 \leq K \leq L$$

$$\begin{aligned} \Delta P &= p_j \max (0, L+t_i-d_j) + p_i \max (0, K-t_j+t_i-d_i) \\ &\quad - p_j \max (0, K-d_j) - p_i \max (0, L+t_i-d_i) \end{aligned}$$

There are three cases which must be considered:

$$\text{CASE I} \quad 0 \leq d_i \leq d_j < L$$

$$\begin{aligned} \Delta P &= p_j (L+t_i-d_j) + p_i \max (0, K-t_j+t_i-d_i) - p_j \max (0, K-d_j) \\ &\quad - p_i (L+t_i-d_i) \\ &= p_j [(K-d_j) - \max (0, K-d_j)] \\ &\quad + p_i [\max (0, K-t_j+t_i-d_i) - (K-t_j+t_i-d_i)] \\ &\quad + (p_j - p_i)(L-K+t_i) + p_i (t_i - t_j) \end{aligned}$$

But ΔP is a sum of terms each not greater than 0 since

$$\text{if } K \geq d_j \text{ then } p_j [K - d_j - (K - d_j)] = 0;$$

$$\text{if } K < d_j \text{ then } p_j [K - d_j - 0] \leq 0;$$

$$[\max (0, M) - M] \leq 0 \text{ for all } M,$$

$$L \geq K \text{ so } L - K + t_i \geq 0;$$

$$p_j \leq p_i \text{ so } p_j - p_i \leq 0; \quad t_i \leq t_j \text{ so } t_i - t_j \leq 0.$$

CASE II

$$d_i \leq L \leq d_j$$

$$\Delta P = p_j \max (0, L + t_i - d_j) + p_i \max (0, K - t_j + t_i - d_i)$$

$$- p_j (0) - p_i (L + t_i - d_i)$$

$$= p_j [\max (0, L + t_i - d_j) - (L + t_i - K)]$$

$$+ p_i [\max (0, K - t_j + t_i - d_i) - (K - t_j + t_i - d_i)]$$

$$+ (p_j - p_i) (L + t_i - K) + p_i (t_i - t_j)$$

But ΔP is a sum of terms each not greater than 0 since

$$K \leq L \leq d_j \text{ and } t_i \geq 0 \text{ so } [\max (0, L + t_i - d_j) - (L + t_i - K)] \leq 0;$$

$$[\max (0, M) - M] \leq 0 \text{ for all } M;$$

$$p_j \leq p_i \text{ so } p_j - p_i \leq 0 \text{ and } L \geq K \text{ so } (L - K + t_i) \geq 0;$$

$$t_i \leq t_j \text{ and } p_i \geq 0 \text{ so } p_i (t_i - t_j) \leq 0.$$

$$\text{CASE III} \quad L \leq d_i \leq d_j$$

$$\begin{aligned} \Delta P &= p_j \max (0, L + t_i - d_j) + p_i \max (0, K - t_j + t_i - d_i) \\ &\quad - p_j (0) - p_i \max (0, L + t_i - d_i) \\ &= p_j [\max (0, L + t_i - d_j) - \max (0, L + t_i - d_i)] \\ &\quad + (p_j - p_i) \max (0, L + t_i - d_i) \\ &\quad + p_i \max (0, K - t_j + t_i - d_i) \end{aligned}$$

But ΔP is a sum of terms each not greater than 0 since

$$d_j \geq d_i \text{ so } L + t_i - d_j \leq L + t_i - d_i \text{ therefore}$$

$$\max (0, L + t_i - d_j) - \max (0, L + t_i - d_i) \leq 0;$$

$$p_j \leq p_i \text{ so } (p_j - p_i) \leq 0 \text{ and } \max (0, M) \geq 0 \text{ for all } M;$$

$$K \leq d_i \text{ and } t_i \leq t_j \text{ so } \max (0, K - d_i + t_i - t_j) = 0.$$

Appendix II

Data Used in the Sample Problem
for Development Stage Testing

Problem 1

Job	1	2	3	4	5	6	7
t_j	3	3	2	1	5	4	4
d_j^j	2	5	6	8	10	15	17
p_j^j	2	6	8	2	4	6	3

Problem 2

Job	1	2	3	4	5	6	7	8	9	10
t_j	80	6	16	49	32	97	61	66	23	12
d_j^j	15	25	31	31	32	55	57	64	67	73
p_j^j	1	1	1	1	1	1	1	1	1	1

Problem 3

Job	1	2	3	4	5	6	7	8	9	10
t_j	4	1	2	4	1	4	2	2	3	2
d_j^j	3	4	7	8	11	15	16	20	20	25
p_j^j	3	1	4	2	3	5	1	5	3	10

Problem 4

Job	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
t_j	2	4	2	3	1	4	2	2	3	3	5	1	6	2	3
d_j^j	3	5	6	8	11	13	16	18	22	26	30	31	36	39	40
p_j^j	1	3	5	2	4	1	3	5	2	4	1	3	5	2	4

Problem 5

Job	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
t_j	3	5	1	4	2	4	5	1	3	2	5	2	3	4	1	5	3	2	1	4
d_j^j	3	6	9	11	14	17	20	22	25	28	31	33	36	39	42	44	47	50	53	55
p_j^j	1	3	5	2	4	1	3	5	2	4	1	3	5	2	4	1	3	5	2	4

Problem 6

Job	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
t_j	4	4	5	5	5	2	1	5	5	3	2	2	4	2	3	5	4	3	5	4
d_j^j	1	6	11	15	21	23	28	29	34	38	39	41	46	50	53	55	59	63	67	69
p_j^j	1	1	4	5	5	1	1	1	5	5	2	1	5	1	6	5	1	6	4	2

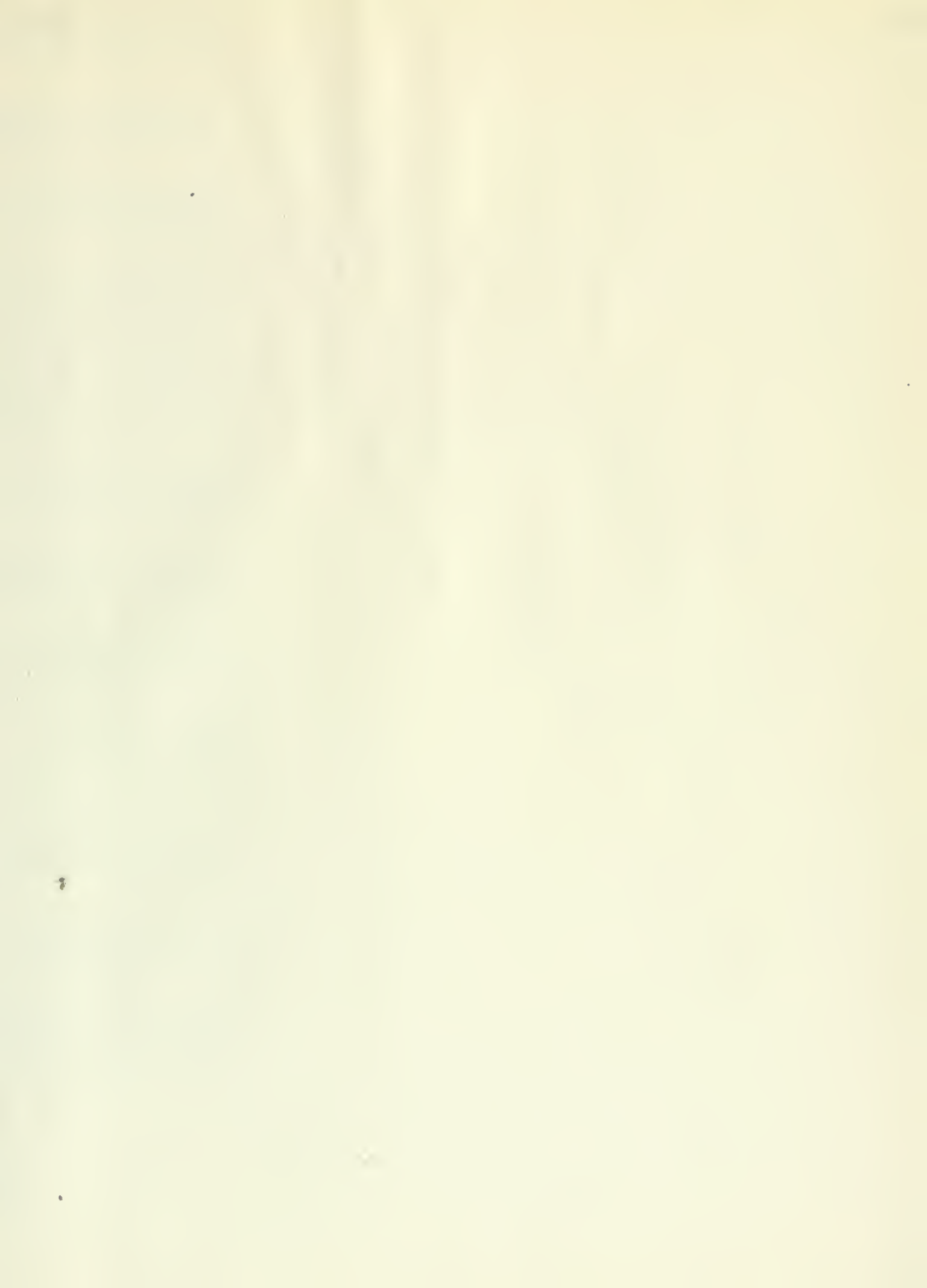
BIBLIOGRAPHY

1. Conway, R. W., W. L. Maxwell, and L. W. Miller, Theory of Scheduling, Addison Wesley, Reading, Mass., 1967.
2. Elmaghraby, S. E., "The One Machine Sequencing Problem with Delay Costs," Journal of Industrial Engineering, Vol. XIX, No. 2 (February 1968), pp. 105-108.
3. Elmaghraby, S. W., "The Machine Sequencing Problem--Review and Extensions," Naval Research Logistics Quarterly, Vol. 15, No. 2 (June 1968), pp. 205-232.
4. Elmaghraby, S. E., private communication, September 8, 1969.
5. Emmons, H., "One-Machine Sequencing to Minimize Certain Functions of Job Tardiness," Operations Research, Vol. 17, No. 4 (July-August 1969), pp. 701-715.
6. Held, M., and R. M. Karp, "A Dynamic Programming Approach to Sequencing Problems," Journal of the Society for Industrial and Applied Mathematics, Vol. 10, No. 1 (March 1962), pp. 196-210.
7. Jackson, J. R., "Scheduling a Production Line to Minimize Maximum Tardiness," Management Sciences Research Project, Research Report No. 43, UCLA, July 1955.
8. Lawler, E., "On Scheduling Problems with Deferral Costs," Management Science, Vol. 11, No. 2 (October 1964), pp. 280-288.
9. Maxwell, W. L., "The Scheduling of Single Machine Systems: A Review," The International Journal of Production Research, Vol. 3, No. 3 (1964), pp. 177-199.
10. McNaughton, R., "Scheduling with Deadlines and Loss Functions," Management Science, Vol. 6, No. 1 (September 1959), pp. 1-12.
11. Moore, J. M., "An N Job, One Machine Sequencing Algorithm for Minimizing the Number of Late Jobs," Management Science, Vol. 15, No. 1 (September 1968), pp. 102-109.
12. Schild, A., and I. J. Fredman, "On Scheduling Tasks with Associated Linear Loss Functions," Management Science, Vol. 7, No. 3 (November 1961), pp. 280-285.
13. Schild, A., and I. J. Fredman, "Scheduling Tasks with Deadlines and Non-linear Loss Functions," Management Science, Vol. 9, No. 1 (September 1962), pp. 73-81.

BIBLIOGRAPHY

(continued)

14. Smith, W., "Various Optimizers for Single Stage Production,"
Naval Research Logistics Quarterly, Vol. 3, Nos. 1 and 2
(March and June 1956), pp. 59-66.



DATE DUE

~~NOV 3 '76~~	AUG 29 '83	
~~FEB 14 '76~~	~~MAR 14 '76~~	
DEC 07 '78	NOV 25 '78	
MAR 3 '77	NOV 25 '78	
MAR 28 '77		
APR 22 '77	FEB 01 1980	
NOV 13 '73	JAN 20 1980	
APR 3 '76		

Lib-26-67

MIT LIBRARIES



453-70

3 9080 003 875 660

MIT LIBRARIES



454-70

3 9080 003 906 663

MIT LIBRARIES



456-70

3 9080 003 906 630

MIT LIBRARIES



457-70

3 9080 003 906 606

HD2

.M41

Nos. 4

Nos. 46

MIT LIBRARIES



458-70

3 9080 003 906 671

MIT LIBRARIES



459-70

3 9080 003 875 652

MIT LIBRARIES



460-70

3 9080 003 906 622

MIT LIBRARIES



461-70

3 9080 003 875 611

MIT LIBRARIES



462-70

3 9080 003 875 637

MIT LIBRARIES



463-70

3 9080 003 875 678

MIT LIBRARIES



464-70

3 9080 003 875 645

MIT LIBRARIES



465-70

3 9080 003 906 580

